AI-OS Autonomous Architecture Guide

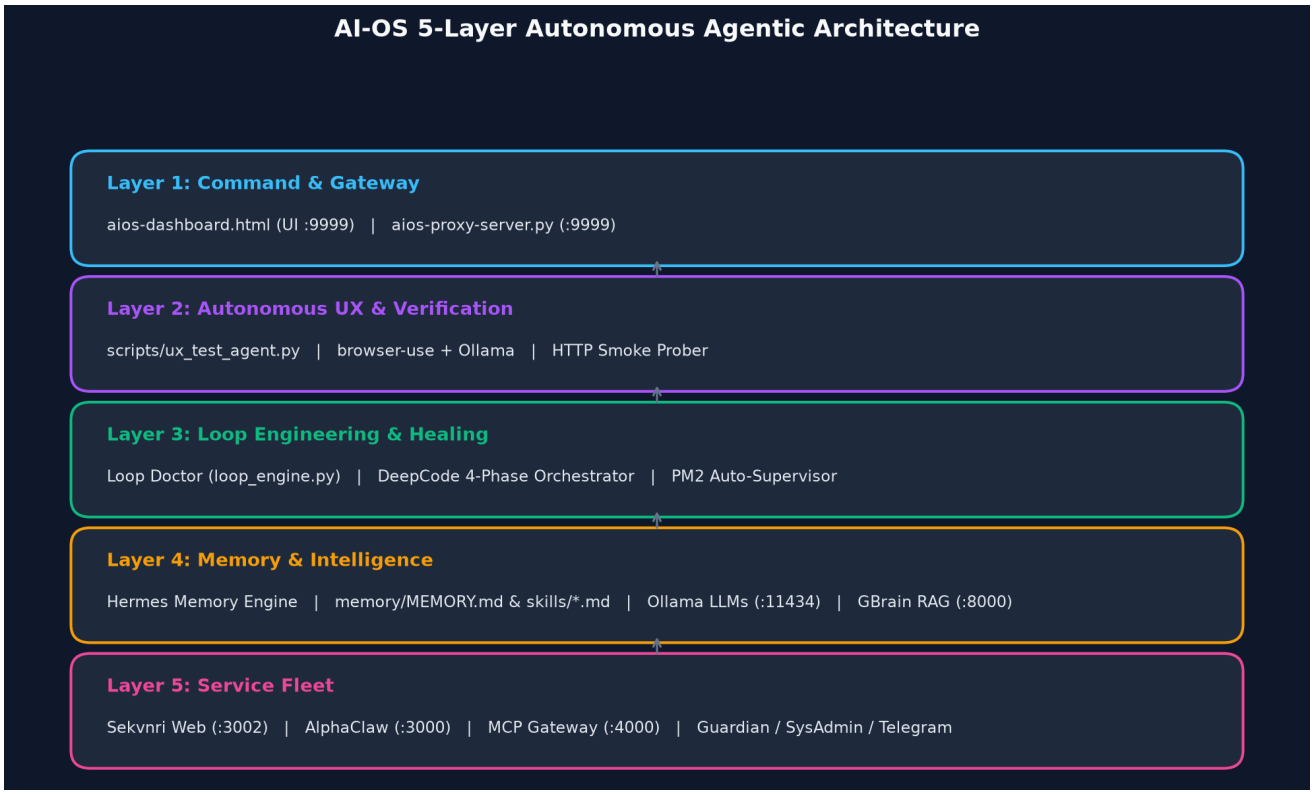
Complete Technical Breakdown, Closed-Loop Self-Healing & Module Interactions

Executive Overview

The **AI-OS** is an entirely local, self-contained autonomous agent operating system. It integrates real-time UI/UX testing agents (powered by **Browser-Use** and local **Ollama** LLMs), a 4-phase loop triage and diagnostic engine (**DeepCode** / **Loop Engineering**), long-term memory and skill synthesis (**Hermes Memory Engine**), and robust process auto-recovery via **PM2**.

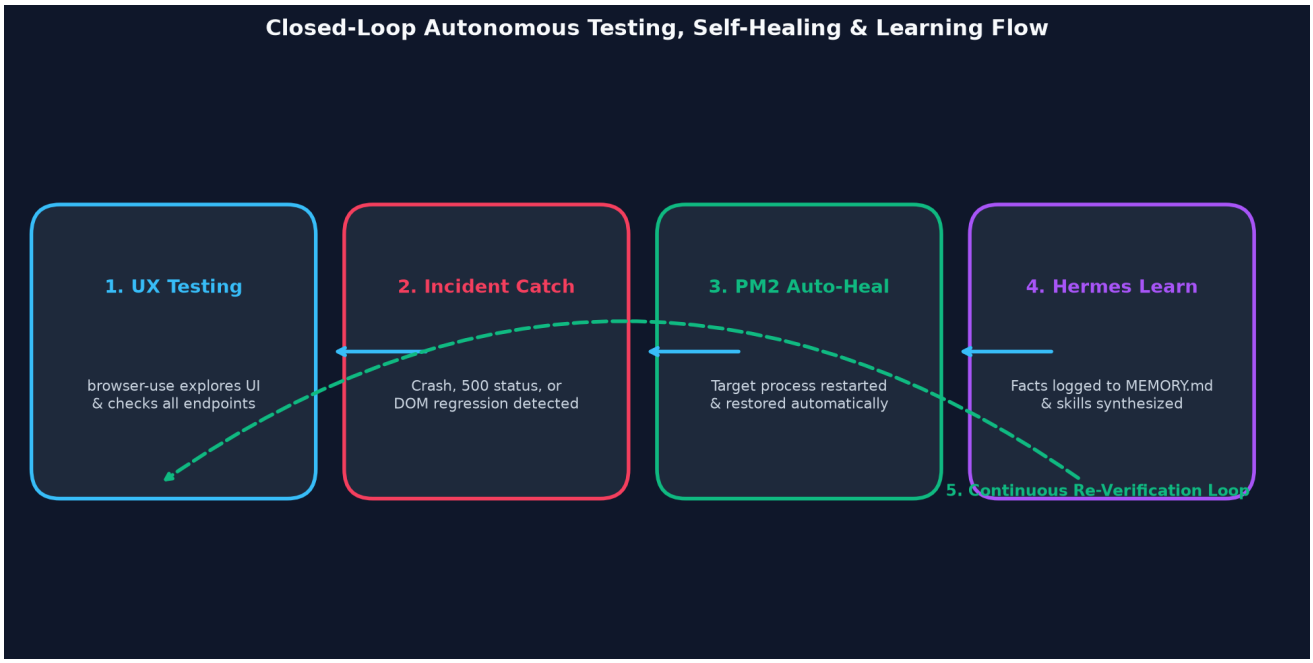
1. System Architecture Layers

The system operates across 5 decoupled layers, guaranteeing zero single points of failure and full local autonomy:



2. Closed-Loop Autonomous Lifecycle & Self-Healing

Unlike traditional monitoring that only raises alerts, AI-OS implements an active **closed-loop recovery pipeline**:



Lifecycle Phases in Detail:

- **Phase 1 (Autonomous Probing):** The UX Agent launches browser sessions or HTTP probes to verify every service endpoint, navigation tab, and interactive form.
- **Phase 2 (Incident Detection):** If a crash, HTTP 500/503, or unhandled exception occurs, the agent catches it and extracts failure metadata.
- **Phase 3 (Self-Healing Intervention):** The agent automatically calls PM2 to restart and restore the degraded process without human involvement.
- **Phase 4 (Memory & Skill Consolidation):** The root cause and fix action are appended to memory/MEMORY.md and synthesized into Markdown SOP skills in skills/.
- **Phase 5 (Re-Verification):** A post-healing test pass verifies the restored service and marks the status as Healthy in the live dashboard feed.

3. Detailed Module Responsibilities

Module / File	Primary Task	Key Autonomous Capability
aios-dashboard.html (Port 9999 / 3000)	Autonomous Command Center UI	Live fleet status grid, CPU/memory stats, 1-click UX test trigger, and action buttons.
aios-proxy-server.py (Port 9999)	Gateway & Control Tower	Routes proxy requests, executes background test tasks, serves /health, and handles self-healing API.
ux_test_agent.py (scripts/)	Autonomous UX & Smoke Agent	Emulates user interactions via Browser-Use + Ollama; automatically restarts failing services.
loop_engine.py (scripts/)	Loop Doctor & Support Triage	Audits system health; runs 4-phase DeepCode pipeline to resolve open support tickets.
hermes_memory_engine.py (mcp-server/)	Long-Term Memory Engine	Extracts learned facts from trajectory logs and synthesizes permanent markdown skills.
GBrain Knowledge Base (Port 8000)	Embedded Vector RAG	Provides semantic context and domain retrieval using embedded PGLite vector storage.

4. Fleet Service Mapping & Port Registry

All services run persistently in local user space and are supervised by the PM2 daemon:

5. Step-by-Step Operator Guide

A. Launching the Command Dashboard

Open your web browser and navigate to **http://localhost:9999**. The dashboard will automatically poll all services every 10 seconds and reflect live health badges.

B. Triggering Real-Time UX Tests

1. Scroll to the ■ **Real-Time UX Agent Testing** section.
2. Select your target service or keep ■ **All Services** selected.
3. Click ■ **Run UX Test**. Results will update in real time with duration and step logs.

C. Command Line Execution (CLI)

Execute tests and doctor diagnostics directly in your terminal:

```
python3 scripts/ux_test_agent.py --target all
python3 scripts/loop_engine.py doctor
python3 scripts/loop_engine.py audit
```

D. Triggering via HTTP API / Webhooks

```
curl -X POST http://localhost:9999/proxy/ux-test -H 'Content-Type: application/json' -d
'{"target":"all"}'
```

End of Technical Guide *— AI-OS Autonomous Operations*